



Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

Compiladores: Análise Semântica

Prof^a Thaína A. A. Tosta

tosta.thaina@unifesp.br

São José dos Campos – 2021/2

Análise Semântica

O objetivo geral da análise semântica é verificar se partes distintas do programa estão coerentes;

Etapa da análise do código apoiada por heurísticas

- ▶ Difícil de ser formalizada por meio de gramáticas
 - ▶ Associada a inter-relacionamentos entre partes distintas do código
- Heurística é um conjunto de regras e métodos que conduzem à descoberta, invenção ou resolução de problemas;
 - A **tabela de símbolos** assume papel fundamental durante a análise semântica.

Análise Semântica

Tabela de símbolos

- Estrutura de dados auxiliar criada para apoiar a análise semântica;
- Essa estrutura normalmente é implementada como uma tabela de *hashing* (tempo constante);
- Conteúdo usual da tabela de símbolos:
 - Nome do identificador
 - Tipo do identificador (variável, função...)
 - Escopo da variável
 - Tipo de dados do identificador (*int*, *float*, *void*...)
 - Número da linha em que o identificador aparece no programa fonte

Análise Semântica

Tabela de símbolos

- Estratégia geral para popular a tabela de símbolos (considerando a estrutura de C-):
 - Pode ser feito durante a análise léxica, sintática ou semântica;
 - Função de *hashing* deve gerar entrada para
 - Nome de função
 - » Calcular *hashing* usando apenas os caracteres do nome da função
 - Nome de variável
 - » Calcular *hashing* usando caracteres do nome da variável + nome da função
 - » Demanda controle do escopo atual

Análise Semântica

Tabela de símbolos

- Estratégia geral para popular a tabela de símbolos
 - Ao encontrar ID, verificar próximo *token*:
 - Se for “(”, então ID é função;
 - Senão, ID é variável;
 - Se *token* anterior for “int”, “float” ou “void”, fazer entrada na tabela de símbolos;
 - Senão, apenas atualizar número de linhas do ID na tabela de símbolos.

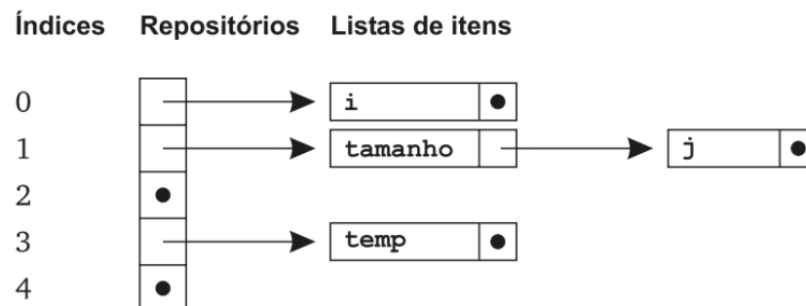
Programa de entrada:

```
1 int gcd(int u, int v)
2 { if (v==0) return u;
3   else return gcd(v, u-u/v*v);
4 }
5 void main(void)
6 { int x; int y;
7   x = input(); y = input();
8   output(gcd(x,y));
9 }
```

Análise Semântica

Tabela de símbolos

- Uma tabela de *hashing* é uma matriz, onde cada linha é um repositório com um índice de inteiros;
- A função de *hashing* transforma a cadeia de caracteres do nome do identificador em um valor inteiro do intervalo de índices;
- Para lidar com colisões (itens colocados em um mesmo índice), o **encadeamento separado** pode ser utilizado com implementação por uma lista linear para cada repositório.



Análise Semântica

Tabela de símbolos

- É necessário definir um tamanho para a tabela *hashing*, utilizando uma atribuição a um número primo;
- A conversão do nome do identificador para um valor inteiro pode ser feita como abaixo, onde h é o índice da tabela *hashing*, n é a quantidade de caracteres do nome do identificador c ;
- Utilizando pesos α , é possível lidar melhor com possíveis colisões.

$$h = (\alpha^{n-1}c_1 + \alpha^{n-2}c_2 + \cdots + \alpha c_{n-1} + c_n) \bmod tamanho = \left(\sum_{i=1}^n \alpha^{n-i} c_i \right) \bmod tamanho$$

Análise Semântica

Principais tarefas realizadas durante a análise semântica estática (em tempo de compilação):

- Verificação de declarações (e escopo);
- Verificação de tipos;
- Verificação da unicidade de declaração de variáveis;
- Verificação de fluxo de controle.

Análise Semântica

- Verificação de declarações

Verifica se as variáveis utilizadas no programa foram devidamente declaradas (quando a linguagem exigir).

```
#include <iostream>
using namespace std;

int main() {
a = 10;
cout << "Valor de a: " << a << endl;
}
```

Mensagem de erro

...cpp: In function 'int main()':

...cpp:5: error: 'a' was not declared in this scope

Análise Semântica

- Verificação de declarações e escopo

```
#include <iostream>
using namespace std;
int main() {
    int a = 9;
    void mostra();
    mostra();
}
void mostra() {
    cout << "a: " << a << endl;
}
```

Mensagem de erro

...cpp: In function 'void mostra()':

...cpp:11: error: 'a' was not declared in this scope

Análise Semântica

- Verificação de tipos

Verifica se as variáveis declaradas estão sendo usadas de forma coerente, de acordo com o tipo especificado.

```
#include <iostream>
using namespace std;

int main() {
    int a = 9;
    float b = 5;
    cout << "a%b: " << a%b << endl;
}
```

...cpp: In function 'int main()':

...cpp:7: error: invalid operands of types 'int' and 'float' to binary 'operator%'

Análise Semântica

- Verificação da unicidade de declaração de variáveis

Detecta duplicações em declarações de variáveis.

```
int main()  
{  
    int a;  
    float a;  
    ...  
}
```

Mensagem de erro

```
In function 'main':  
...: duplicate member 'a'
```

Análise Semântica

- Verificação de fluxo de controle

Detecta erros nas estruturas de controle do programa (*for, do, while, if else, switch case*).

```
void exemplo(int j, int k)
{
    if (j == k) break;
    else continue;
}
```

Mensagem de erro

In function 'exemplo':

...: break statement not within a loop or switch

...: continue statement not within a loop

Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(1)

```
void exemplo()
```

```
{
```

```
    int a; a = 0;
```

```
    b = a;
```

```
    ...
```

```
}
```

“variável não declarada”, considerando que *b* não seja global

Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(2)

```
void exemplo()  
{ ... }
```

```
void main(void)
```

```
{
```

```
    int a;
```

```
    a = exemplo();
```

```
    ...
```

```
}
```

“atribuição inválida”, *a*
é do tipo `int` e
exemplo() não retorna
nada

Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(3)

```
void main(void)
{
    void a;
    ...
}
```

“declaração inválida de variável”, *void* só pode ser usado para declaração de função

Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(4)

```
void main(void)
```

```
{
```


```
    int a;
```

```
    int a;
```

```
    ...
```

```
}
```

“declaração inválida de
variável”, *a* já foi
declarada previamente



Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(5)

```
int fun1() { ... }  
int fun2() { ... }  
void main(void)  
{  
    int a;  
    a = fun3();  
    ...  
}
```

← “*fun3()*: chamada de função não declarada”

Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(6)

```
int fun1() { ... }
```

```
int fun2() { ... }
```

```
int fun3() { ... }
```

“função *main()* não declarada”

Análise Semântica

Principais erros semânticos que devem ser detectados no projeto de C-

(7)

```
int fun1() { ... }  
int xyz() { ... }  
void main(void)  
{  
    int a;  
    int xyz;  
    ...  
}
```

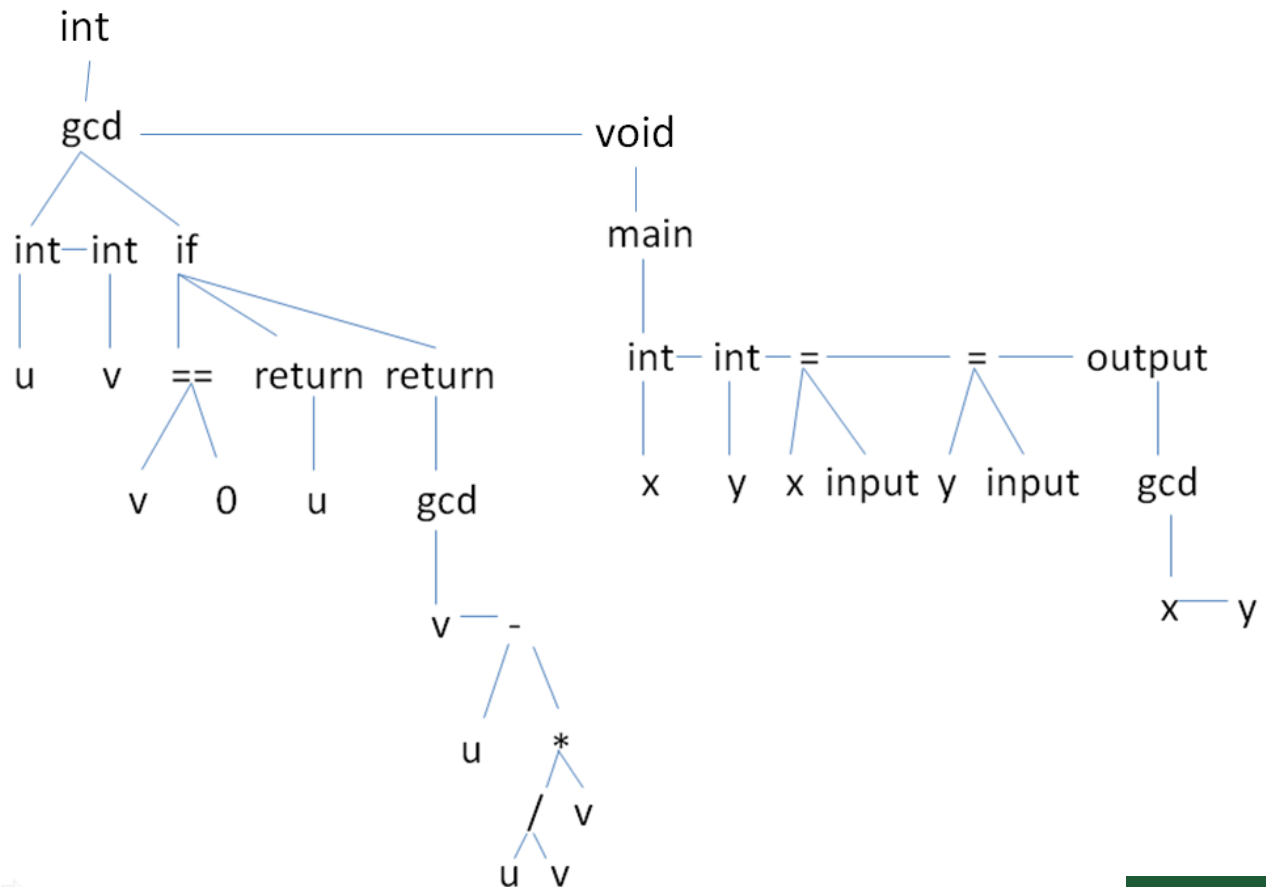
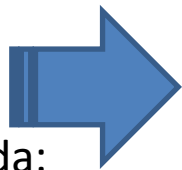
← “declaração inválida”, xyz já foi
declarado como nome de
função

Análise Semântica

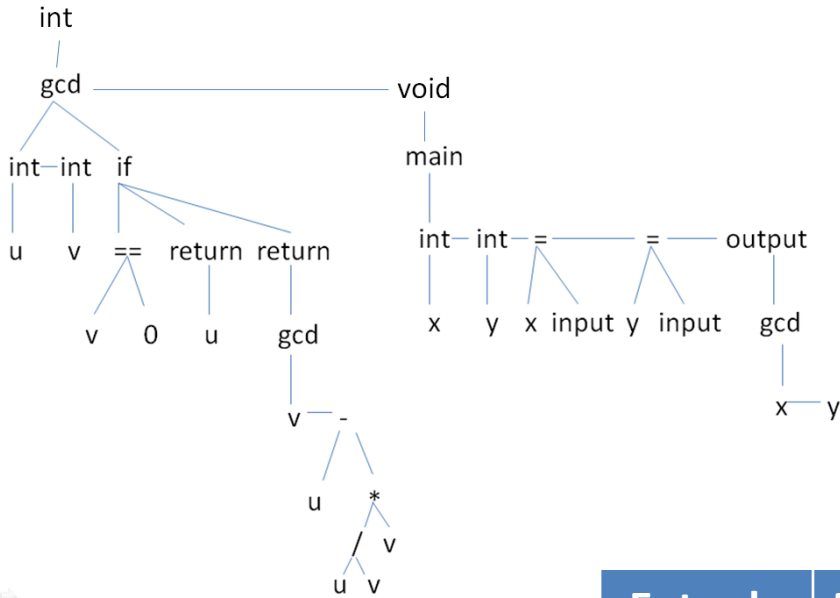
Árvore sintática do programa para cálculo do gcd (Louden, apêndice A)

Programa de entrada:

```
1 int gcd(int u, int v)
2 { if (v==0) return u;
3   else return gcd(v, u-u/v*v);
4 }
5 void main(void)
6 { int x; int y;
7   x = input(); y = input();
8   output(gcd(x,y));
9 }
```



Análise Semântica



Programa de entrada:

```
1 int gcd(int u, int v)
2 { if (v==0) return u;
3   else return gcd(v, u-u/v*v);
4 }
5 void main(void)
6 { int x; int y;
7   x = input(); y = input();
8   output(gcd(x,y));
9 }
```

Tabela de Símbolos

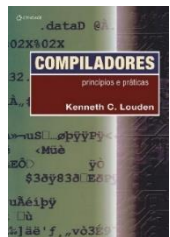
Entrada	Nome ID	Escopo	Tipo ID	Tipo dado	n. Linha
...					
12	u	gcd	var	int	1,2,3
...					
136	main		fun	void	5
...					
200	gcd		fun	int	1,3,8

Análise Semântica

Bibliografia consultada



RICARTE, I. **Introdução à Compilação**. Rio de Janeiro: Editora Campus/Elsevier, 2008. (Cap. 5)



LOUDEN, K. C. **Compiladores: princípios e práticas**. São Paulo: Pioneira Thompson Learning, 2004. (Cap. 6)